Deep Reinforcement Learning in Large Discrete Action Spaces

Maanik, Shashank, Nikhil

1 Abstract

In RL problems, it is essential to be able to reason about large discrete action spaces. There are several problems such as recommender systems and language models where such models would be very useful. Current approaches based on value function approximation require us to perform mod $\mathcal{A}$ (No. of actions) evaluations in order to choose the best action for a given state, which is intractable when $\mathcal{A} > 10000$. The authors embed the actions in a continuous space and leverage an actor-critic based architecture for the given task. They propose a novel training method named Wolpertinger algorithm 2, that first predicts a proto-action $\hat{a}$ using the actor, and uses k-nearest neighbour algorithm to find the k-closest neighbours to $\hat{a}$ in the discrete action set $\mathcal{A}$. The Q values for these k actions are evaluated using the critic network and the best action is chosen based on the corresponding score.

They show the efficacy of their method on 3 different environments: 1) Discrete Continuous Environments, 2) Multi-Step Planning Environment & 3) Recommender Systems. The model performs better than the baseline in all the cases and experiments reveal that the approach is robust to changes in the k-nearest neighbour algorithms and small values of k are better for training with respect to computational time.

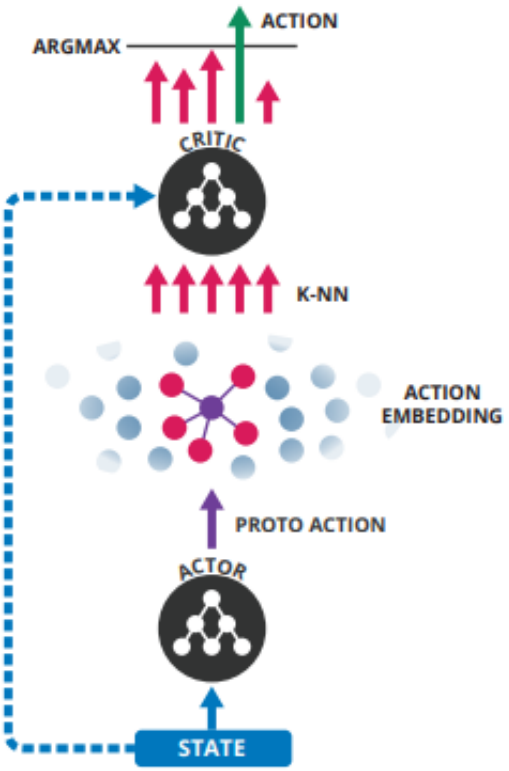


Figure 1: Wolpertinger Architecture

2 Introduction & Approach

There are two main types of RL approaches: 1) Value Function Approximation & 2) Policy Gradient algorithms. In the case of value function approximation algorithms, the policy is value based. Usually, the policy is a greedy policy relative to the value function as shown below:

$$\pi_Q = \arg\max_{a \in \mathcal{A}} Q(s, a)$$

The issue with such value based policies is that when the action space is very large, such as a million actions, a total of $|\mathcal{A}|$ critic evaluations are necessary. This approach is thus, intractable for such large action spaces. The policy based approaches on the other hand, parameterize the policy as $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$. These methods avoid the computational cost of evaluating the Q-values but do not generalize well over the action space. Thus, this paper proposes an actor-critic approach that is Sub-linear complexity relative to the action space and is able to generalize well over the action space.

Algorithm 1 Wolpertinger Policy

State $\mathbf{s}$ previously received from environment.
$\hat{\mathbf{a}} = f_{\theta^\pi}(\mathbf{s})$ {Receive proto-action from actor.}
$\mathcal{A}_k = g_k(\hat{\mathbf{a}})$ {Retrieve k approximately closest actions.}
$\mathbf{a} = \arg\max_{\mathbf{a}_j \in \mathcal{A}_k} Q_{\theta^Q}(\mathbf{s}, \mathbf{a}_j)$
Apply $\mathbf{a}$ to environment; receive $r, \mathbf{s}'$.

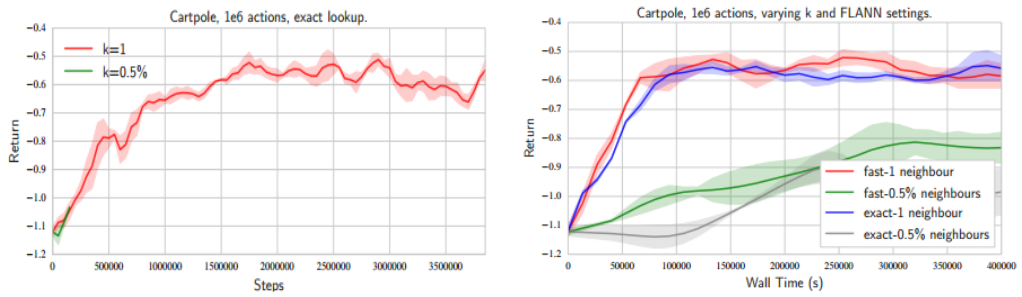
Figure 2: Wolpertinger Training

2.1 Wolpertinger Training

The authors propose a Wolpertinger training algorithm as shown in Figure 2. The actions and state are approximated using function approximators such as neural networks and the k-nearest neighbour searching is done in logarithmic complexity using FLANN algorithm [3]. The main steps of the algorithm are given below:

- **Action Generation**: The actor network generates a proto-action $\hat{a}$ in the continuous action space that may or may not belong to the discrete action set $\mathcal{A}$. An approximate k-nearest neighbour algorithm such as FLANN [3] is then applied to get k actions in set $\mathcal{A}$ that are closest to the proto-action $\hat{a}$. The k actions are chosen such that they minimize the L2 norm with the proto action: $g_k(a) = \arg\min_{a \in \mathcal{A}} |a - \hat{a}|_2$
- **Action Refinement**: However, simple nearest neighbour search ($k = 1$) cannot be used as it is possible that the nearest neighbour of $\hat{a}$ in $\mathcal{A}$ may not have a high Q-value. Thus, the authors use a small number of $k(> 1)$ and rank the actions based on the Q-value scores predicted by the critic network. The action with highest Q-value is then considered as the best action.
- **Policy Gradient**: Training is done by simply following the policy gradient of the actor network ($\hat{a}$) instead of the action finally chosen.
- **DDPG**: The optimization is done using DDPG [1] algorithm. Their approach also uses Replay buffers and target networks, two approaches to improve training stability that were introduced in [2].

The above algorithm is more scalable than the previous algorithms such as DDPG [1] because we perform evaluations (forward pass on critic) for only k actions instead of all the actions belonging to $\mathcal{A}$.



(a) Value of K

(b) FLANN Algorithm used

Figure 3: Results of the algorithm on the inverted pendulum environment

3 Experiments

They test their algorithms on 3 different types of environment that are described in brief as below:

- **Discretized Continuous Environment**: We used the MuJoCo physics simulator to simulate the classic continuous control tasks *cartpole*. Each dimension d in the original continuous control action space is discretized into i equally spaced values, yielding a discrete action space with $|A| = i^d$ actions. The environment is used to demonstrate that the proposed training algorithm is able to reason in both the small and large action spaces.
- **Multi step plan environment**: These environments has i actions available at each time step and an agent needs to plan n time steps into the future then the number of actions i^n is quickly intractable for arg max based approaches. The above task is implemented on a puddle world environment which has 2 base $actions : \{down, right\}$. This means that environments with a plan of length n have 2^n actions in total, for $n = 20$ we have $2^{20} \approx 10^6$ actions. Thus, the environment demonstrates agents ability to discern problems with very large action spaces.
- **Recommender Environment**: Such tasks demonstrates agents performance on real world tasks involving very large action spaces. The authors constructed an environment named *Recommender System* to utilizing live large-scale recommender system. Again, the results on such task shows agents ability to reason with very large action spaces.

4 Results & Analysis

Apart from showing better results on the large discrete action space environments such as the 3 mentioned earlier, the paper also does a lot of ablation studies on the effect of different parameters such as k-nearest neighbour algorithm, the number of nearest neighbours (k) and approximate vs exact algorithms. The details of these ablation parameters are given below:

- **K nearest algorithm**: They test their approach with three different types of FLANN algorithms: 1) Slow, 2) Medium, 3) Fast.
- **Number of neighbours**: They test their approach with different number of nearest neighbours such as $k = 1$ (Nearest Neighbour), $k = 0.05|\mathcal{A}|$ (0.5% of action space) and $k = |\mathcal{A}|$ (All actions).
- **Approximate vs Exact Algorithms**: They tried two variants of nearest neighbour algorithms, 1) Approximate algorithms such as FLANN and 2) Exact algorithms (Evaluate L2 norm with all actions to find k-nearest neighbours).

5 Results

Figure 3 shows the performance of the algorithm on the inverted pendulum problem. As mention earlier, this environment was discretized uniformly into 1 million actions for training and testing. The figures clearly indicate that the algorithm is able to solve the given environment despite the large size of the action space. The same is true for the other 2 environments as well, however, we do not report them here as they are already explained in detail in the paper.

The most interesting aspect of the analysis is the ablation studies on k and algorithm types as mentioned earlier. Based on the plots, we can infer the following:

- The value of k (Number of nearest neighbours) significantly affects the performance of the model. If the value of k is very high, then the training time of the model is too high and the model isn't able to train well as shown in Figure 3(a). For example, if $k = 0.05|\mathcal{A}|$, then the algorithm completes only 100k steps instead of the 3.5 million steps for $k = 1$ in the same amount of time. However, experiments on other environments show that $k = 1$ itself is not sufficient, thus, it is best to have a small number of k (¿ 1) such $k = 5$.
- From 3(b) it is clear that using approximate algorithms such as Fast FLANN is much faster as compared to exact algorithms and give much better returns for the same duration of training time.
- From 4, we can infer that the Fast FLANN is much faster than the exact algorithms and the medium and slow FLANN. However, the type of algorithm used does not matter if the value of k is very high such as $0.05|\mathcal{A}|$ as shown in row 2 of Figure 4.

# Neighbors	Exact	Slow	Medium	Fast
1	18	2.4	8.5	23
0.5% – 5,000	0.6	0.6	0.7	0.7

Figure 4: Median steps/second as a function of k & FLANN settings on cart-pole

6 Our Approach

As the given algorithm is similar to DDPG, we implement DDPG algorithm as well as WOLPERTINGER algorithm. To test our approach, we experiment on the Continuous cartpole environment provided by OpenAI Gym (Custom environment). We convert this continuous action environment into a discrete environment by dividing the continuous action space (-1, 1) into a million discrete actions in (0, 1). The link to our code and presentation is given below:

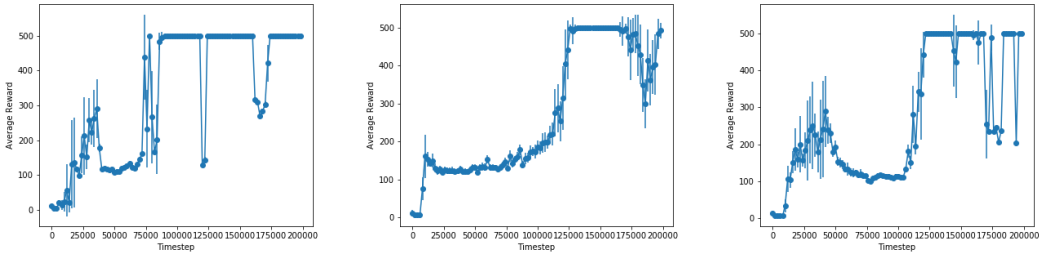
- Code: `https://rebrand.ly/toml-project-code`
- Presentation: `https://rebrand.ly/toml-presentation`
- Video: `https://rebrand.ly/toml-video`

We implement all our models and code using PyTorch. For k-nearest neighbour algorithm, we use the PyFlann library that implements several logarithmic k-nearest neighbour algorithms. All experiments were done on a CPU based machine only and no GPU was used. Our code is based on two main codes that are available online. Two main codes that we used for our implementation are:

- jimkon: Unofficial implementation of the paper using Tensorflow
- DDPG: Implementation of DDPG given in a medium article.

6.1 Our results

We show the results of our approach below. Similar to the experiment methodology in the paper, we try our approach with multiple values of k. We specifically try $k = 1$, 10, 1000 & 100000. The plots for the same are shown in Figure 5. From the plots, we can see that our model is able to learn the dynamics of the environment very well and at the end of around 150,000 steps, it consistently has a mean reward of 500, which is the maximum reward possible in this environment. From 5, we can also notice that training is fastest in the case of $k = 1$, suggesting that for this task, it is the most suitable option of all the three. However, for some other tasks, $k = 1$ might not be enough and we might need to use slightly higher value of $k = 10$ in such cases. Another advantage of using $k = 10$ is that, its reward plot is the most smooth out of all the three plots as shown in Figure 5(a), 5(b) & 5(c). We also experiment with $k = 100000$ but the model did not train at all in this case as the computation time was too high, suggesting the efficacy of our approach.



(a) K = 1 (b) K = 10 (c) K = 1000

Figure 5: Results of our code on the Discretized Continuous Cartpole Environment

We also evaluated the various values of k for computational time by running the training loop for a maximum of 20,000 steps each. The results for the same are shown in Table 1 below. From Table 1, we can see that $K = 1$ is just marginally faster than than $K = 10$ but significantly faster than $K = 1000$. This suggests that we should use $K = 10$ in general as it might work better in other environments as compared to $K = 1$ and is not much slower in terms of computations times as well.

Number of neighbours (K)	Time Taken (minutes)
1	4.37
10	4.45
1000	6.24
100000	$>$ 5 hours

Table 1: Time taken for 20,000 steps

7 Work Division

The division of work amongst team-mates is as follows:

- Shashank:
  - Wrote the code for converting action space from continuous to discrete
  - Contributed to DDPG code
  - Contributed the most to report
- Nikhil:
  - Wrote the WOLPERTINGER training code
  - Contributed to DDPG code
  - Contributed briefly to report & presentation
- Maanik:
  - Contributed to DDPG code
  - Wrote the plotting code and ablation studies
  - Contributed the most to presentation

8 Conclusion

The authors propose a new training method called WOLPERTINGER algorithm for reinforcement learning tasks with very large discrete action spaces. We implement their approach in PyTorch for the Discretized Continuous Cartpole environment and show that the model is able to reason in case with very large action space such as $|\mathcal{A}| = 10^6$. We also perform ablation studies on the different values of k such as 1,10 & 1000 and show that the model performs well in all of these cases but $k = 1$, 10 is computationally much more efficient.

References

[1] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[3] Marius Muja and David G Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence*, 36(11):2227–2240, 2014.